Fundamentals of Databases

Notes for CS 6400: Database Systems Concepts & Design
Georgia Tech (Dr. Jay Summet with Dr. Leo Mark Christensen), Summer 2017
as recorded by Brent Wagenseller

Overview

- We will learn
    - what a database (DB) is
    - what a database management system (DBMS) is
- What is a database / a database management system?
    - A **database** (**DB**) is a model of reality
    - A **database management system** (**DBMS**) is a software system allowing users to create and maintain a database
- 3 Major topics of this course
    - Data modeling
    - Process Modeling
    - Database efficiency

Models

- Models can be useful when we want to examine or manage part of the real world
- A **model** is a means of communication
    - <u>The users of a model must have a certain amount of knowledge in common</u>
        - For example, 'north is up'
    - Is described in some language
    - <u>A model only emphasizes selected aspects of reality, namely those that are useful for the purpose at hand</u>
    - Features may exist that do not exist in reality
- The costs of using a model are often considerably lower than the costs of using or experimenting with the real world itself
- Examples of models
    - A **map** is a type of model
    - A model of the US economy
    - Storm warning systems
    - Traffic simulation
- A point of contention about models: it may be hard to agree on how the model is built!
    - There may be different opinions about colors, orientation, scale, etc
- Models can have errors, and this can be a problem!
- Models can have an abundance of information
    - This can lead to information overload
    - This can also lead to information that is not helpful for some circumstances

- Models can save time, if the alternative is learning from the environment yourself
    - Think of reinforcement learning – it takes a long time for an agent / ‘robot’ to learn its environment, but with a map it could quickly get to its destination without having to take precious time exploring the world

Functionality of a Database Management System (otherwise known as: “When to Use a Database Management System”

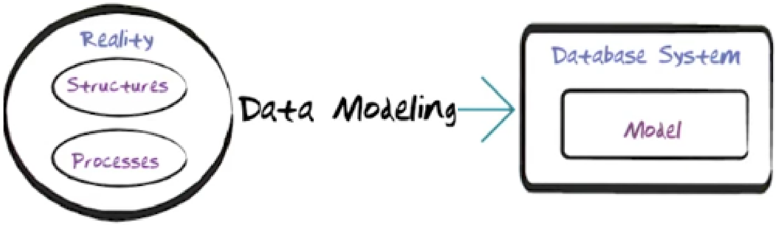
- If the application is data intensive, use a DBMS
    - **Data intensive** applications is where a lot of data may flow between users and a database
    - Note its specified that the alternative is **process intensive**, which is hinted as the alternative to data intensive
- These are the general properties / functions of a DBMS
    - (These are all listed so they are important)
    - Persistent storage of data
        - Even after a shutdown!
    - Centralized control of the data
        - Its possible to enforce policies
    - Redundancy can be controlled
        - In this sense, **redundancy** means controlling consistency and integrity
            - This will be discussed later!
        - Consistency means you cannot derive contradiction / inconsistencies from the database
    - Multiple user support
        - **Matter of fact, sharing data is the key behind supporting communication!**
    - Data documentation for the structure of data
    - Data independence
        - That is to say, the ability to change the implementation of the database without having to change the user interface
    - Control over the security / access to the data
    - Backup / Recovery is possible with a DBMS

When Not to Use a Database Management System

- If the initial investment in hardware, software, and training is too high
- If the generality is not needed
    - For example, we may not need the security, concurrency, control, or recovery
- If the data is too simplistic and / or stable
- If real time requirements cannot be met by it (that is, the DB would slow the application down)
- Multiple users are a non-issue

Data Modeling

- Again, the **model** represents a perception of structures of reality



- The **data modeling** process has two steps:
    - Fix a perception of structures of reality
        - The extended entity relationship model is good for this (more on that later)
    - Represent this perception
        - The relational relationship model is good for this (more on that later)
- In the data modeling process, we select aspects of reality that are important, and then we abstract to organize these aspects

Process Modeling

- In **process modeling** we aim at fixing and representing a perception of processes of reality
    - As opposed to the structures of the database, the processes are not represented in the database
        - Rather, they are reflected in how we use the database through the DBMS
        - It seems like process modeling has to do with the actual, literal SQL queries that interact with the database – via Perl, Python, PHP, etc
- BRENTS NOTE: The term **data manipulation language** (**DML**) is used here; a more familiar term may be SQL query, although it seems the professor wishes to keep the discussion abstract
    - That said, the professor does state we will exclusively be using SQL; so while there may be other languages, SQL is the dominant language and the one we will be using
- There are two ways we can use the database through the DBMS
    - Embedded in program code
        - Example: SQL hard-coded in Perl, Python, PHP; any GUI that accepts input and the pre-canned DML incorporates it in the background
    - Executed ad-hoc
        - Example: SQL written at the command prompt, sqlyog, mysql-workbench, Toad, PL-SQL
            - This seems to entail actually writing the DML / SQL by hand

Data Models, Database Architecture, and Database Management System Architecture

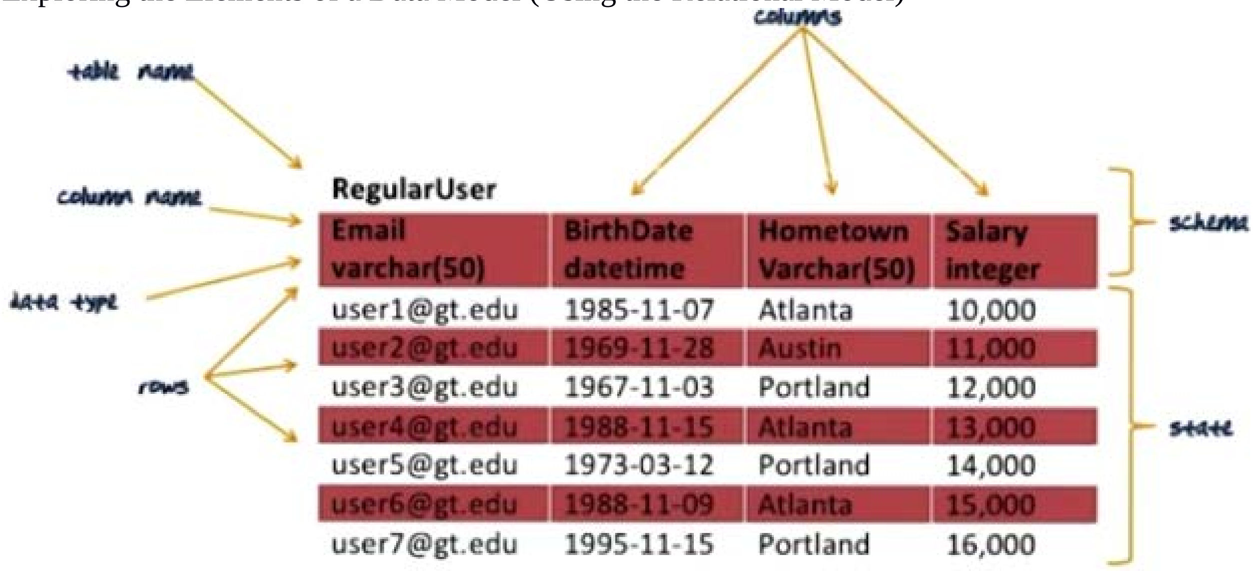
- Data modeling requires a tool called a **data modeler**
- Data modelers offer formalisms to express
    - Data structures

    - Constraints
    - Operations
    - Keys and identifiers
    - Integrity and consistency
    - Null values
    - surrogates
- In addition, the architecture of a DBMS is important, as it allows for the creation of a data model too
- Major points about the architecture of a DBMS:
    - Database
        - ANSI/SPARC 3-level DB Architecture
            - This is what we will be using in this class
    - Data independence DBMS

Example of Data Models

- <u>A data model is NOT the same as a model of data!</u>
    - A database is a model of structures that represent reality
    - A data model is the <u>tool</u> / formalism that we use to create such a model
- There are three main elements a data model handles
    - Data structures
    - Integrity constraints
    - Operations
- Three examples of data models
    - **Entity-relationship model**
        - This will be used to <u>fix a perception of reality</u> in this course
    - **Relational model**
        - This will be used to <u>implement</u> the entity-relationship model above
    - **Hierarchal model**
        - This was the first data model used by IBM IMS DB in 1967
        - It is the underlying model for XML

Exploring the Elements of a Data Model (Using the Relational Model)



RegularUser

Email varchar(50)	BirthDate datetime	Hometown Varchar(50)	Salary integer
user1@gt.edu	1985-11-07	Atlanta	10,000
user2@gt.edu	1969-11-28	Austin	11,000
user3@gt.edu	1967-11-03	Portland	12,000
user4@gt.edu	1988-11-15	Atlanta	13,000
user5@gt.edu	1973-03-12	Portland	14,000
user6@gt.edu	1988-11-09	Atlanta	15,000
user7@gt.edu	1995-11-15	Portland	16,000

columns
table name
column name
data type
rows
schema
state

- <u>Data Model Elements: Data Structures</u>
    - In the relational model, data is represented in tables
        - BRENTS NOTE: Think of it like a tab on an Excel spreadsheet
    - The table has a name – in this case its RegularUser
    - The columns
        - The columns have names too: Email / BirthDate / Hometown / Salary
        - The columns also have a data type – character / datetime / character / integer
            - NOTE: varchar(XX) means ‘a string of variable length up to XX’
        - The number of columns is called the **degree** of the table
    - The rows
        - There are a variable number of rows in the table; in this case there are 7 rows
        - A row represents a single entry in the table
    - The schema
        - The **schema** is the collection of:
            - The table name
            - The column names
            - The column data types
        - The schema is stable over time! It is not expected to change!
    - The **state** of the database is represented by the rows of the data
        - They are dynamic and are <u>expected</u> to change over time!
    - The schema represents the structure of the data, but the rows represent the current state of reality that is modeled by the table
    - Relational model constraints

- Data Model Elements: Constraints
  - **Constraints** represent rules that cannot be expressed by the data structures alone
  - Examples (using data above)
    - Email must be unique
      - This will ensure unique rows
    - Emails are not allowed to be null
    - Birth dates must be after 1900-01-01
    - Hometown must be cities in the US
  - Notice that none of these constraints can be represented by the table structure or the data types- this is why constraints express rules that cannot be expressed by the data structure!
- Data Model Elements: Operations
  - BRENTS NOTE: It seems operations are commands that can be issued to the data model to change the state of the model (i.e. add or remove rows)
    - Note that its also possible to change the data structure itself (add columns, change data types, etc)
  - Examples
    - Change state operation: INSERT
      - The data model allows for inserting rows into a table
      - BRENTS NOTE: The INSERT command is a well known command in the relational model (possibly others)
      - Example in psudocode (using data structure above)

```
INSERT INTO RegularUser (Email, BirthDate, Hometown, Salary)
VALUES (‘user11@gt.edu’, ‘1992-10-22’, ‘Atlanta’, 12500);
```

    - Retrieve state operation: SELECT
      - Example in pseudocode:

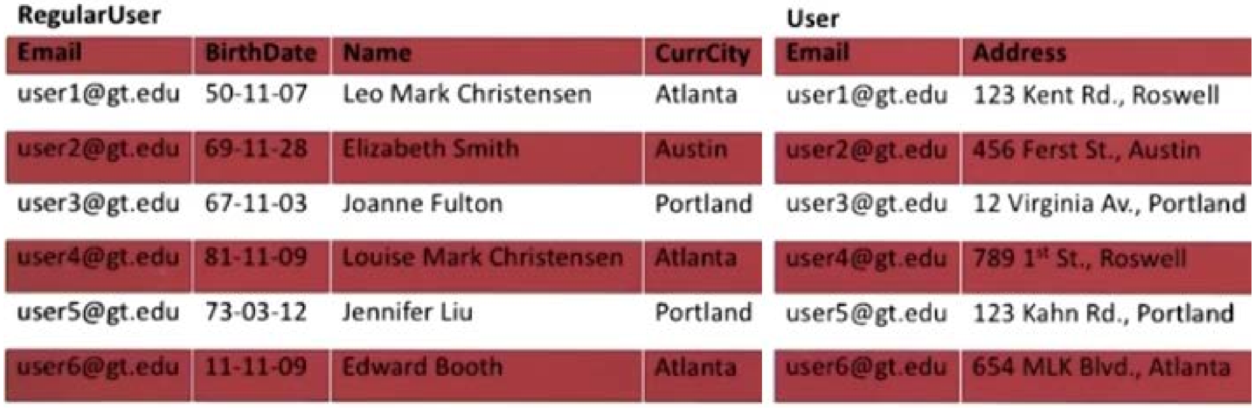
        ```
        SELECT Email, BirthDate
        FROM RegularUser
        WHERE Hometown = ‘Atlanta’ AND Salary >12000
        ```

      - This statement is an operation the retrieves information from the data model
      - There are only two columns selected – the other two are not shown in the returned data (although the do still exist, just not displayed)
      - Note that there are **conditions** put in place via the WHERE clause: the home town of the row must be Atlanta and the salary must be greater than 12000
- Keys
  - **Keys** are uniqueness constraints
  - Keys are either a single column – OR a combination of multiple columns – that uniquely identify a row

    - Typically they are barred from being NULL
    - Example
        - In the example above, we can make the email the key which would force all emails to be unique in the table RegularUser

Integrity and Consistency

- **Integrity** answers the question: Does the database reflect reality well?
    - Think of it as accuracy / data quality
- **Consistency** answers the question: Is the database free from internal conflicts?
    - Think of it as similar columns having different values
- Consistency Example:



RegularUser

Email	BirthDate	Name	CurrCity
user1@gt.edu	50-11-07	Leo Mark Christensen	Atlanta
user2@gt.edu	69-11-28	Elizabeth Smith	Austin
user3@gt.edu	67-11-03	Joanne Fulton	Portland
user4@gt.edu	81-11-09	Louise Mark Christensen	Atlanta
user5@gt.edu	73-03-12	Jennifer Liu	Portland
user6@gt.edu	11-11-09	Edward Booth	Atlanta

User

Email	Address
user1@gt.edu	123 Kent Rd., Roswell
user2@gt.edu	456 Ferst St., Austin
user3@gt.edu	12 Virginia Av., Portland
user4@gt.edu	789 1st St., Roswell
user5@gt.edu	123 Kahn Rd., Portland
user6@gt.edu	654 MLK Blvd., Atlanta

    - Notice that in the table 'RegularUser', users 1 and 4 are listed as in Atlanta, but in the 'User' table they are listed as in Roswell
    - This inconsistency means the database cannot be used to answer the question 'where does the user live'
    - The problem is there is redundancy in the data entry: BOTH tables allow for the entry of a city. When the SAME data can be entered in MULTIPLE spots – like city above – this opens the door for users to erroneously enter data inconsistently
- Integrity is mostly centered around the accuracy of the data
    - Think of it as data quality
    - Dr. Leo gave the example of user #4 – his daughter – is sometimes confused for male as her middle name is 'Mark'

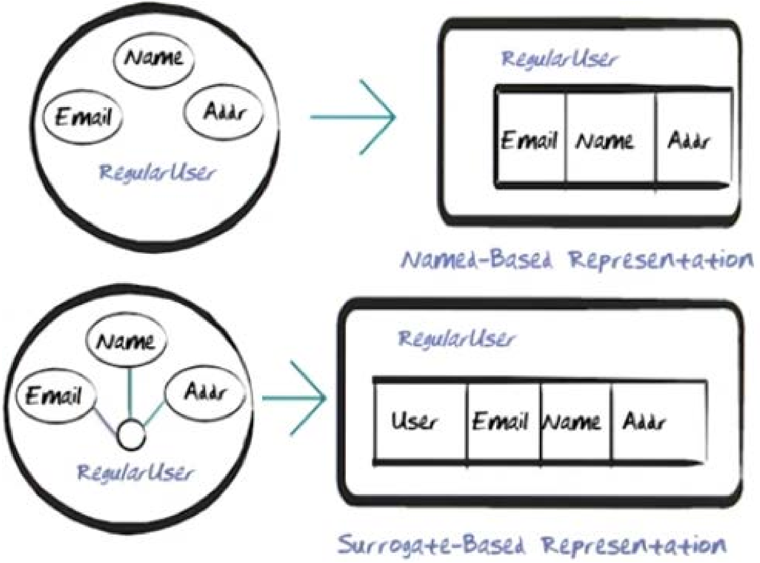
NULL Values

- **NULL** means 'unknown'
- NULL values are important in tables as its quite possible a value is unknown and we need to represent this
- Note that NULL for 'inapplicable' is not advisable – if its 'inapplicable' the data in the table is not grouped properly. In theory, there should be no column in a table that is 'inapplicable' to ANY row/record!

    - If there are ‘inapplicable’, it means either
        - The row does not belong in that table
        - The column does not belong in that table
    - This can happen when a **catch-all form** is used; basically, it’s a single form that is filled out by everyone, if the data is applicable or not.
        - This is convenient for filling out forms, but is a no-no when storing the data in the backend database; official database rules say this data should reside in a separate table

Surrogates

- **Surrogates** are system-generated, unique, internal identifiers
- Surrogates are often used as keys for tables
- When a surrogate identifier is used
    - naming conflicts (two ‘John Smith’ entries, for example) are more easily resolved
    - other ‘unique’ identifiers can be changed
        - often, ‘unique’ identifiers can change (like names (for married women) as email addressed
        - using a surrogate frees up these columns so they can be changed without corrupting the integrity of the database
- Name-based vs surrogate-based representations
    - Here is an example of a name-based table and then a surrogate-based table:

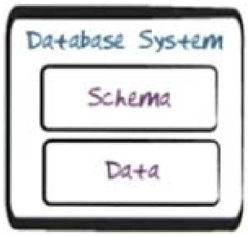


    - In a **name-based representation**, each row is identified by what is known about the row; if it represents people, it’s most likely a name
        - This can lead to problems
    - In a **surrogate-based representation**, a surrogate represents a user outside in the real world
        - Surrogates <u>will not</u> change

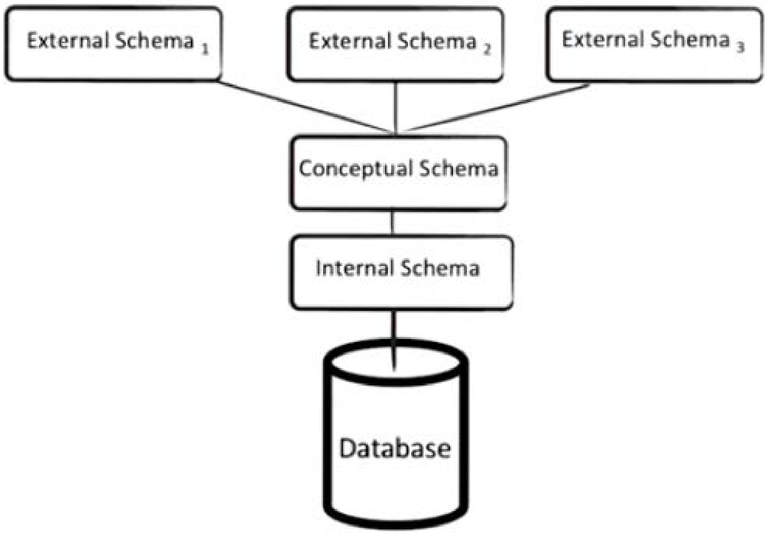
        - Allows for updates of ALL other info attached to the row (which a name-based representation does NOT allow)
- In a surrogate-based representation, regardless of what we know about something and what the values are, the base 'thing' still exists regardless

ANSI/SPARC 3-Level DB Architecture: Separating Concerns

- Recall that a database is a model of structures of reality, and its divided into schema and data:



    - The schema describes the intention (types)
    - The data describes the extension (data)
    - This separation of the schema and data makes the overall system effective and efficient
        - For example, if we want to search for data, we write a query against the schema, and then data is returned that fit the structures of the schema
- In a nutshell, the ANSI/SPARC representation is:



- The creators of ANSI/SPARC 3-Level architecture recognizes that there are aspects that clearly belong in the schema – like how the data is stored and organized – that should NOT be able to be set
    - These aspects – how the data is stored and organized – is known as the **internal schema**
    - Typically these aspects cannot be set by users
    - How it generally works is the request goes through the schema, is sent to the internal schema, the data is retrieved and is then sent back to the schema

- The separation of schema and internal schema has benefits
    - If the internal schema is optimized / upgraded, there is no need to alter the schema
- The above can be further broken down to three parts
    - Three parts
        - External schema
        - Conceptual schema
        - Internal schema
    - The external schema is for the use of the data
        - The **external schema** describes part of the information in the conceptual schema in a form convenient to a particular user group's view
        - The external schema is derived from the conceptual schema
        - There are usually multiple external schemas
        - Many external schemas are needed as different applications may need the data displayed or presented differently
        - BRENTS NOTE: In the lecture video 'External Schema', Dr. Leo quite plainly describes an external schema as a view (he even displays a 'CREATE VIEW' SQL command)
            - A **view** is a way to re-shape a table without actually altering it. This sounds a bit strange, but basically its 'write a query, give the query a name – now this query will look like a table – although its not – but you can treat it as such.'
            - For example, lets say we have a table AAA with columns BBB CCC and DDD; we could create a view ZZZ like

```
CREATE VIEW ZZZ AS SELECT CONCAT(BBB, CCC, DDD) XYZ FROM AAA
```

- So the query above simply concatenates BBB, CCC, and DDD and stores the result as a temp column 'XYZ'; since we saved it as a 'view' ZZZ, we can now simply say

```
SELECT XYZ FROM ZZ
```

- This will be the same as saying 'SELECT CONCAT(BBB, CCC, DDD) XYZ FROM AAA', but it is far more concise
- Note that the original table AAA is not altered in any way – its just a different way of looking at it!
- How are views helpful
    - If the query is complex – say, a hundred+ lines long – you can store the query as a view and every time you reference it in the future it becomes FAR easier to reference (just reference the view)
    - If you believe that column names (or table names) may change in the future, you can write (and then reference) the view instead
        - when it comes time to change your view, all you have to update is the

view – any other user using the view will not have to make any changes at all; any code you may have will not have to be changed at all

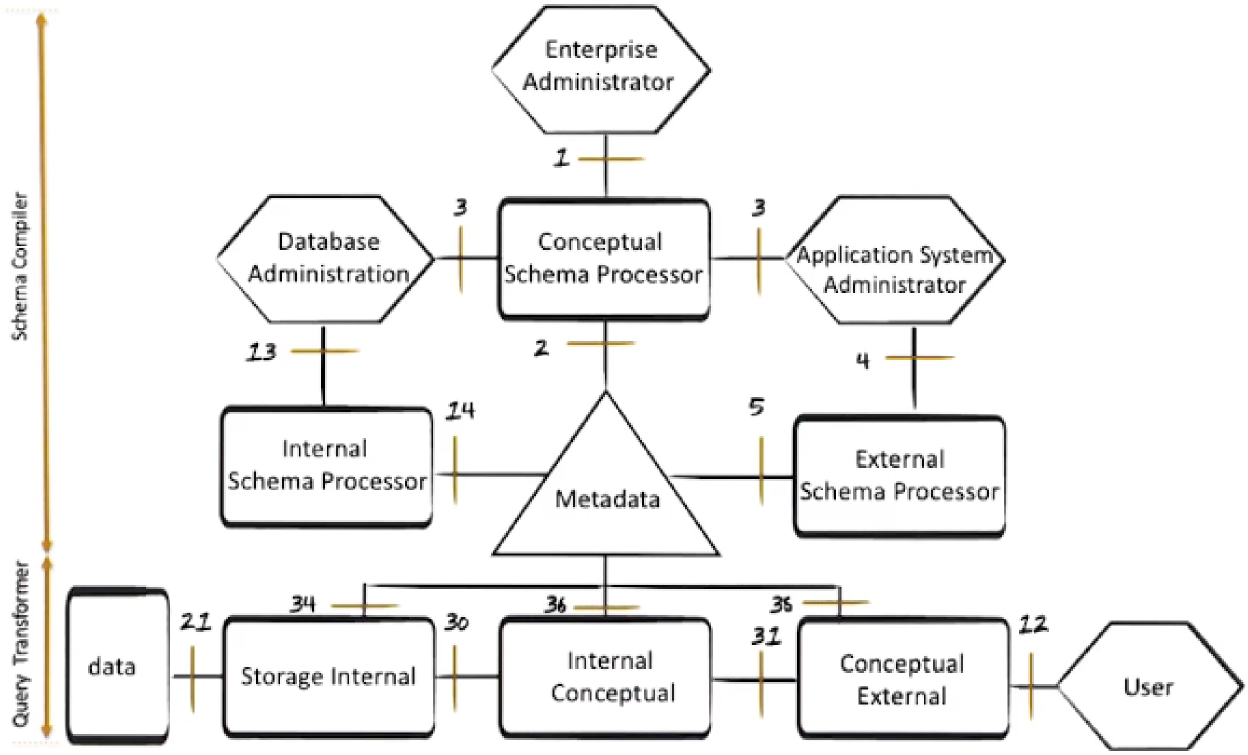
    - The conceptual schema is for the meaning of the data
        - The **conceptual schema** describes all conceptually relevant, general, time-invariant structural aspects of reality
            - BRENTS NOTE: This seems to be the idea of a ‘schema’ we have talked about thus far in the course
        - This is the intermediate idea of a schema
        - It describes nothing related to database representation, physical organization, access, or use
        - The only thing that is visible is the table and the columns
            - So it can see a SQL query, like:

          SELECT Email, BirthDate, MaidenName FROM RegularUser WHERE Sex = ‘F’ and Salary > 70000
            - It cannot include anything about how the results are displayed (other than order) or how the data is accessed
    - The internal schema is for the storage of the data
        - The **internal schema** describes how the information described in the conceptual schema is physically represented to provide the overall best performance
        - Indexing (which will be described later, but its basically a way to ‘speed up’ searching / matching data) is handled in the internal schema
        - BRENTS NOTE: Technically, the conceptual schema cannot see / does not care about the internal schema, but its performance is impacted if the internal schema is not configured properly
- Requests are sent from the external schema, which is translated to the conceptual schema, which is translated to the internal schema; data is accessed and the response is translated up and presented to the application in the format it needs
- Data Independence
    - **Data independence** is concerned with the stability of one layer in the ANSI-SPARC architecture when another is changed
    - The 3 levels if the ANSI-SPARC architecture provides two levels of data independence:
        - Physical data independence
        - Logical data independence
    - **Physical data independence** is a measure of how much the internal schema can change without affecting the application programs
        - Similar to object-oriented programming and encapsulation where the implementation of a class can be changed without affecting the applications that use it
        - Physical data independence is concerned with the separation of the internal

        schema and the conceptual schema

    - Logical data independence is a measure of how much the conceptual schema can change without affecting the application programs
        - Logical data independence is concerned with the separation of the external schema and the conceptual schema
        - Its harder to provide logical data independence (as opposed to physical); this is because if the table changes, it may affect the application

ANSI-SPARC DBMS Framework



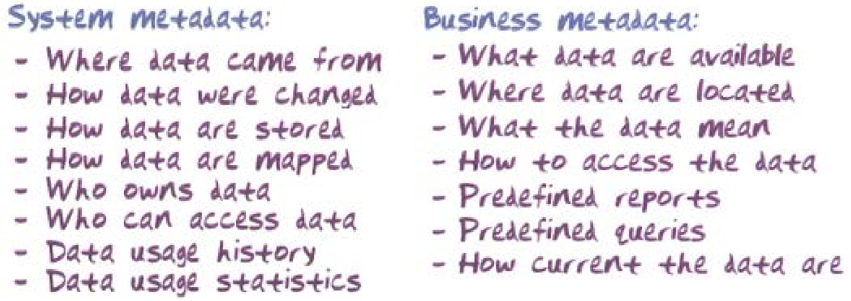
- This is a description of the Database Management System architecture that’s necessary in order to create and support a three-level database
- Note that the different numbers represent either a language or an interface; its an abstract way of defining how the two endpoints either communicate or issue commands
- This was proposed in 1975, before there were any commercial implementations or a relational database
- The framework consists of two pieces
    - The Schema Compiler
    - The Query Transformer
- The Schema Compiler
    - BRENTS NOTE: Note the three distinct (horizontal) tiers inside the schema compiler; these are grouped in this set of three on purpose
    - The hexagons represent people working in different roles with the DBMS

  - The boxes represent process or pieces of software that transform text
  - The triangle represents the metadatabase where schema definitions are stored
  - The role of the **Enterprise Administrator** is to define conceptual database schemata
  - The **conceptual schema processor** ensures the enterprise administrator is using correct syntax
    - If correct, the conceptual schema processor stores the conceptual schema in the metadatabase
  - An **application systems administrator** is responsible for defining external schemata
    - And application systems administrator can investigate the conceptual schema and then create external schemas
  - The **external schema processor**
    - ensures the application systems administrator is using correct syntax
    - it also checks that its correctly logically derived from the conceptual schema
    - If everything is OK, it will store the external schema to the metadatabase
    - Multiple external schema can be defined from the same conceptual schema
  - A **database administrator** can look at the conceptual schema that has been defined for the database and can define an internal schema
  - The internal schema processor
    - passes the internal schema definition, ensuring its syntactically correct
    - makes sure it actually physically implements and supports what was defined in the conceptual schema
    - the internal schema processor stores the internal schema to the metadatabase
- The Query Transformer
  - Regular users (those writing queries) use the interface #12 (between 'user' and 'conceptual external'
    - Two methods
      - Ad hoc queries where the user is actually writing the query themselves
      - A form (say, on a web page) where the user is still interacting with the database but not directly
  - BRENTS NOTE: Note that the communications between User/Conceptual, Conceptual / Internal Conceptual, and the Internal Conceptual / Storage Internal were similarly described: take the communication from the previous node, check the Metadata, and translate it for the node down the line
  - The link between the Storage Internal and the Data is essentially the operating system, and 'language 21' is OS calls
  - After the call to the data, the response traverses all nodes back to the user
  - <u>Note that this is only a theoretical layout; while the functionality of the layout absolutely exists, if all of these internal calls were made the database would be extremely inefficient</u>

- Note this is how it works in theory, but in practice the nodes are combined

Metadata

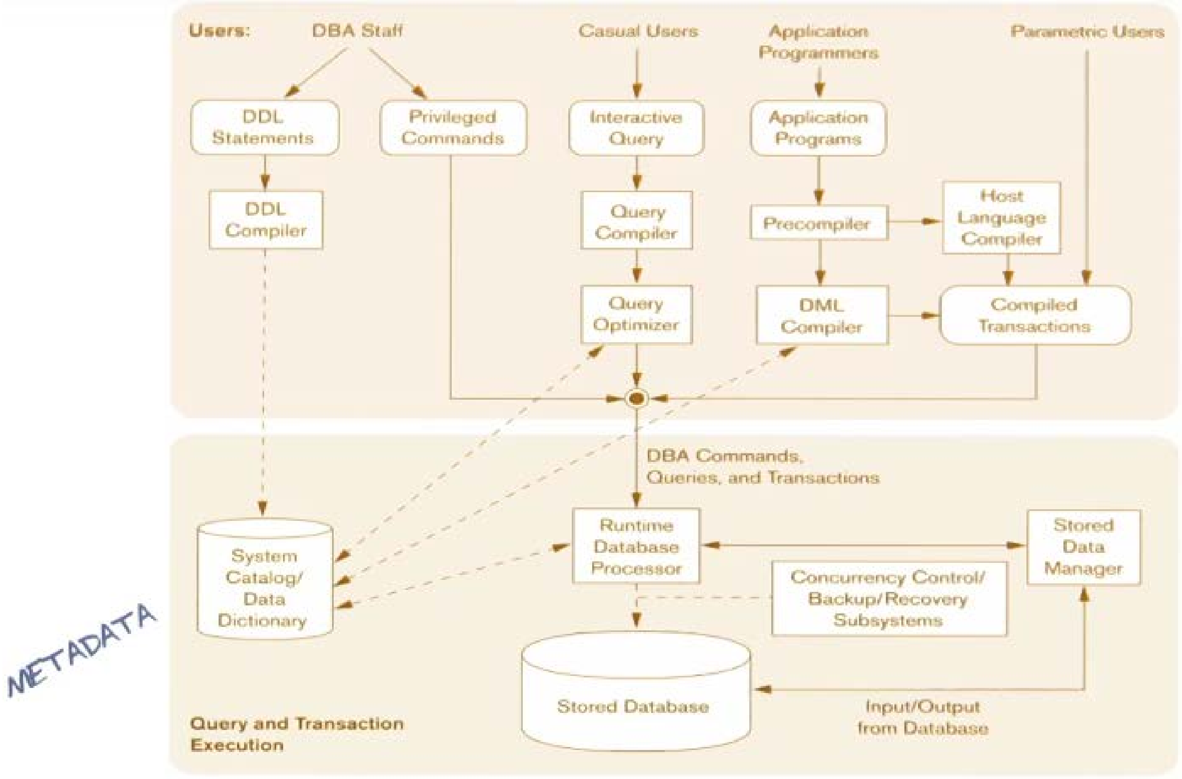
- There are two kinds of metadata
  - System metadata
    - System metadata is critical in a DBMS
  - Business metadata
    - Business metadata is critical in a data warehouse!
- BRENTS NOTE: It seems the difference between the two is the 'system metadata' is a bit more general to databases, whereas the 'business metadata' is more specific to the company / entity using it
- Below is a chart detailing the two:



System metadata:
- Where data came from
- How data were changed
- How data are stored
- How data are mapped
- Who owns data
- Who can access data
- Data usage history
- Data usage statistics

Business metadata:
- What data are available
- Where data are located
- What the data mean
- How to access the data
- Predefined reports
- Predefined queries
- How current the data are

- Metadata Chart



- BRENTS NOTE: For now, not much detail was given on the chart outside of reviewing the flow. This may be expanded upon in further lectures. Any additional notes not represented in the flow I relate below
- This is similar to the previous chart
    - The ‘**DBA Staff**’ are the three roles mentioned before – Enterprise / Database / Application System Administrators
- The **DDL** is the ‘Data Definition Language’, and is used by the ‘DBA Staff’ to perform their tasks
- The **DDL Compiler** compiles the statements created by the DBA Staff and stores it in the metadatabase
- At this point, a **casual user** can write queries that pass through a **query compiler** and query optimizer (which optimizes the query for effectiveness)
    - The query is then given as a command to the **Runtime Database Processor** which will execute it
    - BRENTS NOTE: The ‘Application Programmers’ tree is very similar to the ‘Casual Users’ tree; not much more information was given outside of
- The **concurrency control subsystem** ensures that all competing queries are executed in the proper order
    - This is critical, as there are multiple users at once